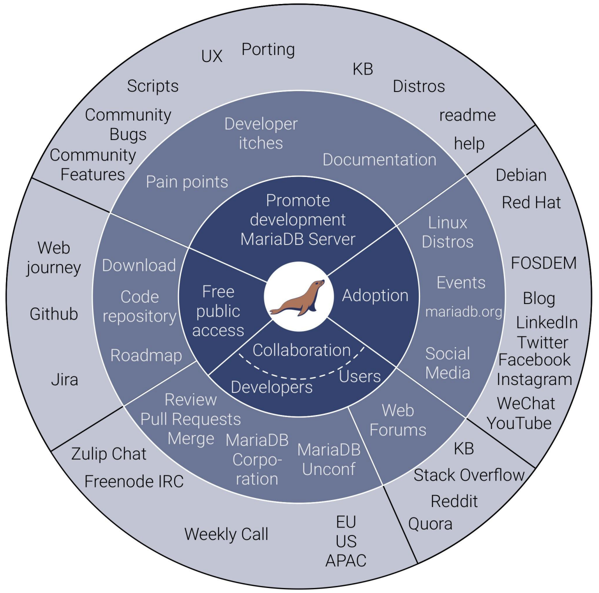


MariaDB Foundation

MariaDB Vector: Making AI transparent on your own data

Use case: Wikipedia editing with RAG

foss-north, Göteborg, Sweden
Fri 14 Apr 2025

Robert Silén
robert.silen@mariadb.org
Community Advocacy, MariaDB Foundation

Why transparency on AI on your own data?

- When accuracy, sources and accountability matter for the user
- When you need control and insight to intermediate steps

How: Use an open source database for your AI apps:

- Store all data in the same open source standard RDBMS: data, sources, vectors and intermediate results
- Single queries can combine vector & standard data
- Developers know the standard RDBMSes
- Numerous tools work with the standard RDBMSes
- Lack of vendor lock in
- Lower total cost

Agenda

1. Use case: RAG for editing Wikipedia

2. What exactly **is** MariaDB Server?
3. What **AI functionality** does MariaDB have?
4. What are the main **use cases** for MariaDB Vector?
5. Steps in **creating a RAG** with MariaDB Vector
6. So **where is the advantage** of using MariaDB?

7. **Technical details** (another TOC; on your own time)

What exactly is MariaDB Server?

Definition: MariaDB is a ***mature extended fork of MySQL***.

- It's near **plug-in compatible** with MySQL
- It's **fully** open source
- It's more **performant**
- It adds plenty of **functionality** on top (sql_mode=Oracle)

Compare MariaDB to:

- **MySQL** (but no vector indexing in Open Source)
- **PostgreSQL** (PG Vector! but slower, plug-in)
- **Vector** databases (but specialised)
- **Oracle** Database (but no vector contender)

What AI functionality does MariaDB have?

Vector indexing and search: We store and search vectors!

- You decide how **you create** the vectors (outside MariaDB)
- We **store** and **index** your vectors (now HNSW)
    - any data (text, images, videos, sound)
- We **search** your vectors (Euclidean, Cosine, soon pushdown)
    - **Nearest-neighbour** search
- **Combining** the above with all the existing other data
- **Pushdown** WHERE conditions: combine search criteria

This is exactly what a standard database is **supposed** to do!

What are the main use cases for MariaDB Vector?

Classic RAG: Retrieval-Augmented Generation

- Do a “specialised ChatGPT”, which gives answers only based on your own data
- Implement this with high relevance (“quality”) and at reasonable cost

Smarter generic search: In online stores and elsewhere

- “You may also be interested in this product”
- Give best guess responses despite vague text input
- Can be combined with exact push-down conditions

RAG - Context for AI answers

1. Preparation Phase: Ingestion

Break content into chunks

→ Convert to vector with **embedding model**

→ Store in **vector DB**

2.1 Query-Time: Retrieval

Convert user query to vector embeddings

→ **Search nearest neighbors**

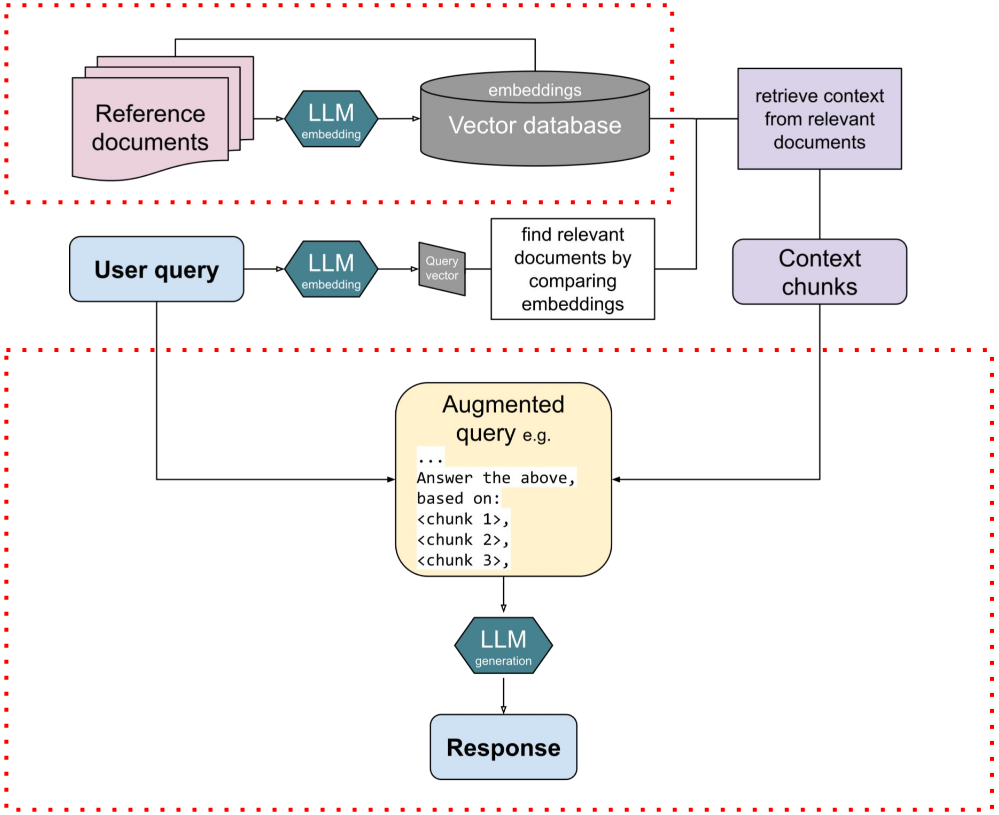
→ Retrieve relevant text chunks

2.2 Query-Time: Answering

Combine query + retrieved context

→ Send as one **text prompt** to language model

→ Generate answer



Steps in creating a RAG with MariaDB Vector 1/2

<u>Assumption</u>: You want to create a specialised ChatGPT

1. You have a <u>mass of text</u> (articles, documentation)
2. The app should answer <u>free-form</u> questions using <u>that text</u>

<u>Step one</u>: Embed the text mass, in a batch run (not “train”)

3. Identify the proper <u>chunk size</u> (unit to index)
4. Create and <u>store the chunks</u> in MariaDB (using eg. Python)
5. <u>Vectorise</u> the chunks (using your favourite LLM)
6. Store and <u>index the vectors</u> (using MariaDB), including “classic” database fields that point to the keys of the chunks in source data

Steps in creating a RAG with MariaDB Vector 2/2

Step two: Run-time, answer the user's free-form question

1. Vectorise the user question (using the same LLM)
2. Search the vector index for the top five-ten nearest neighbours to the vectorised user question (in MariaDB)
3. Concatenate the text chunks of the neighbours into an adequately sized text (in Python)
4. Ask the LLM using the user question as prompt and the concatenated text as context

Voilà: You have provided the user with an answer to a question, based **only** on your own data, at a **low token cost**

Prompt template

```
You are tasked to answer a question using only the following
information:
  [chunk1], [chunk2], [...]
This is the question for you to answer:
  <User query>
Answer with explicit chunk metadata (pages, chapters, url)
```

Prompt example

```
System prompt: You are a Wikipedian-bot that uses only given sources to improve
wikipedia articles. You follow general principles of Wikipedia. You will need the
following material to complete the task below:

<source>
<metadata>
Type: book
Author: Maria Vainio-Kurtakko
Title: Ett gott parti : Scener ur Ellan de la Chapelles och Albert Edelfelts liv
Year: 2022
Publisher: Svenska litteratursällskapet i Finland
Chapter: {name of chapter}
Pages: {x-y}
</metadata>
<source_chunk> {e.g. specific section about Louis Pasteur source.}
</source_chunk></source>

<wikipedia_article url=https://sv.wikipedia.org/wiki/Louis_Pasteur>
<content> {Content of whole article, or part of article} </content>
</wikipedia_article>

<task_instructions>
Using only facts from the given material, suggest one additional new paragraph to
the given Wikipedia article in the same language as the Wikipedia article.
Remember to write from the perspective of the article. Suggest what existing
header to add the new paragraph to, and a new header for this content. Give three
suggestions, with each being half as long as the previous.

Reply without explanations in JSON format with a list of dicts with values
'header_existing', 'header_new' and 'paragraph'. A suitable beginning of each
paragraph could be '{person's surname} was {nature of relationship} with Albert
Edelfelt'
</task_instructions>
```

So where is the advantage of using MariaDB?

Enable and audit the intermediate steps

1. Debug your chunkification; verify intermediate results
2. Test various chunkification strategies

Reuse the index

3. Once every user query
4. Make a summary, an analysis

Save cost

5. Do all the searches on your own database
6. Minimise the expensive usage of AI tokens (“words”)

Further slides: Technical Details

- 16-19 What are vectors
- 20-25 Two-dimensional nns picture (nearest neighbour)
- 27 **CREATE TABLE** .. MariaDB syntax example
- 28 **Python syntax** example
- 29-36 Schemas illustrating RAG
- 38 MariaDB **function** syntax
- 39 **How to download** MariaDB Vector https://mariadb.org/download
- 40 **Benchmarks**: MariaDB Vector is fast!
- 32 Deep dive: Index hierarchy
- **Docs**: https://mariadb.org/projects/mariadb-vector/

MariaDB Vector Integration Frameworks

AI framework integrations

- LangChain, MariaDB Vector Store - python
- LangChain.js, MariaDB Vector Store - node.js
- LangChain4j, MariaDB Embedding Store - java
- LlamaIndex, MariaDB Vector Store - python
- Spring AI, MariaDB Vector Store - java
- VectorDBBench - benchmarking for vector databases

Potential future integrations

- AutoGen - agent to agent
- DSPy - workflow
- Feast - machine learning (not GenAI)
- LangGraph - agentic workflow
- MCP (Model Context Protocol) - integration to external data sources and tools
- Open WebUI - AI Interface
- Google IDX template for MariaDB - visit link to vote for suggestion

Thank you!

Contact details:

robert.silen@mariadb.org

About:

https://mariadb.org/projects/mariadb-vector/

https://sv.wikipedia.org/wiki/Wikipedia:Projekt_Fredrika/SLS-AI-pilot

What is an embedding model vs generative model?

- ChatGPT is a **generative** model.
  - It takes a prompt.
  - Generates the most likely "correct" sequence of words as response.

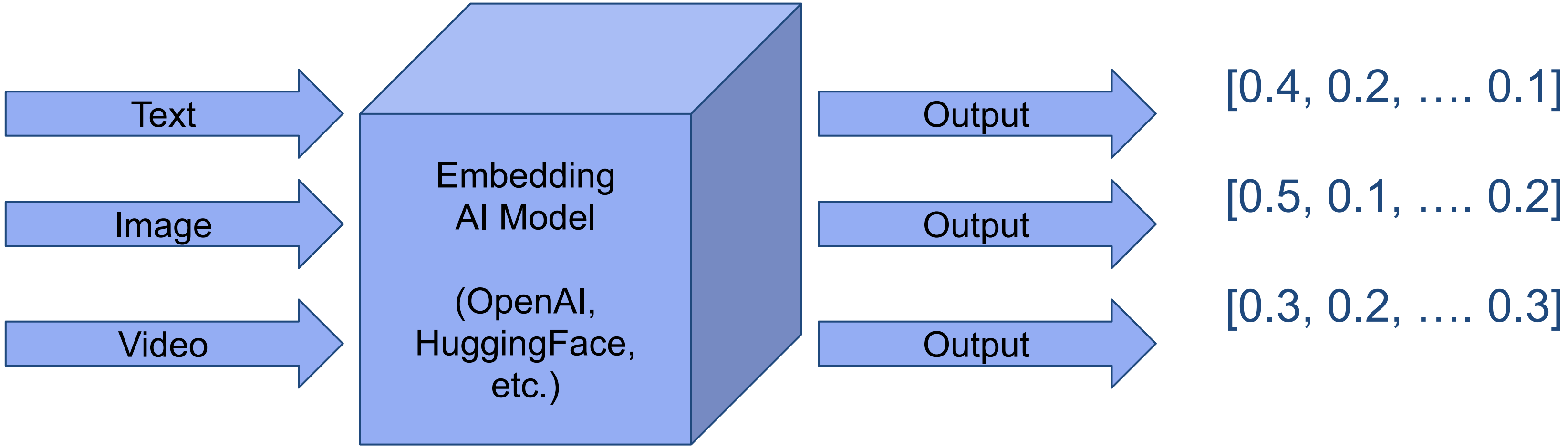
- An **embedding** model generates a vector embedding for a particular prompt.

What is a Vector Embedding?

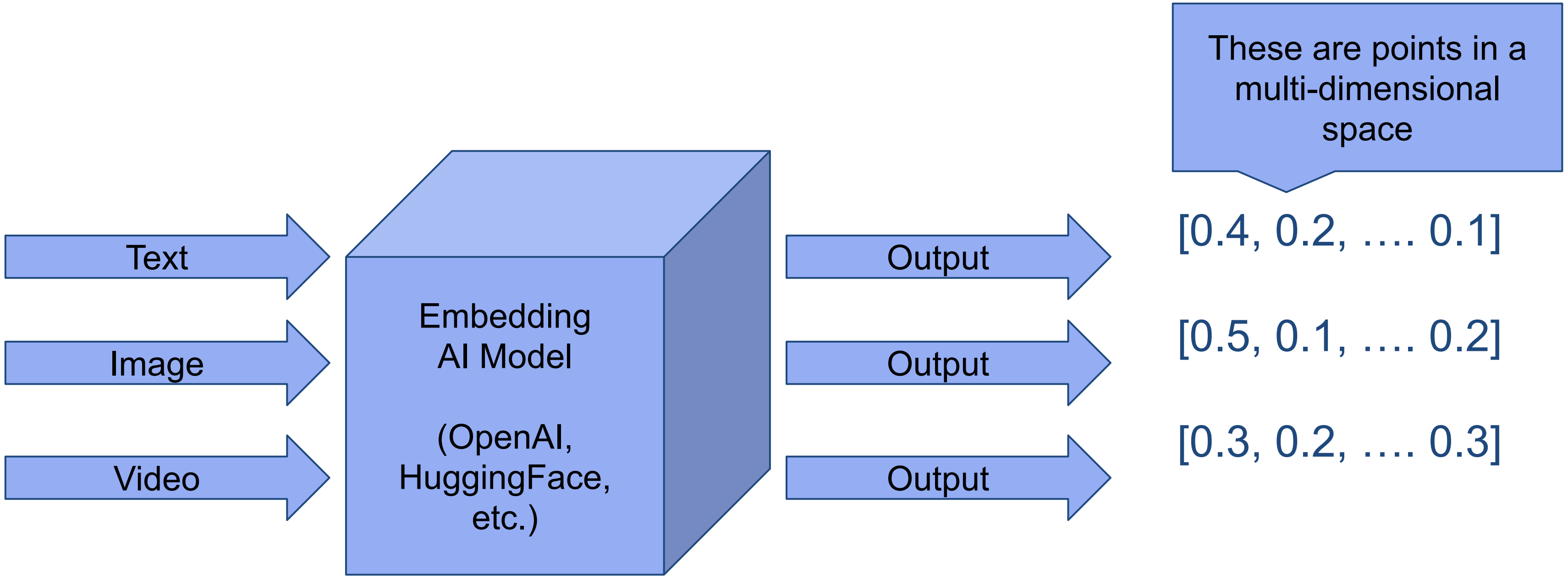
Simply a list of numbers (that describe “features” of the original)

What is a Vector Embedding?

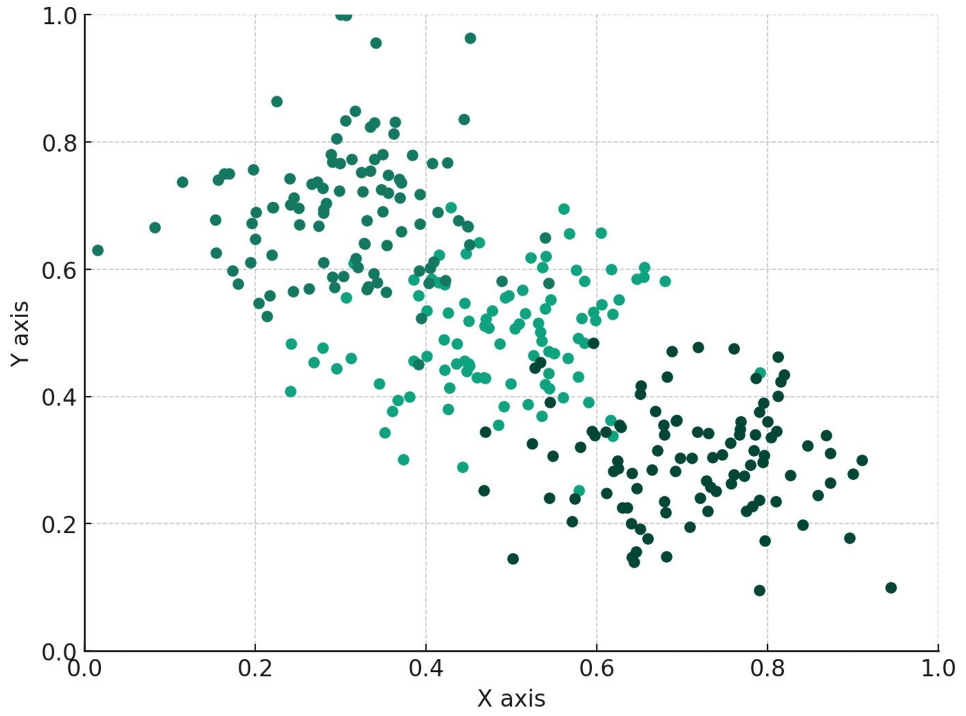
Simply a list of numbers (that describe "features" of the original)



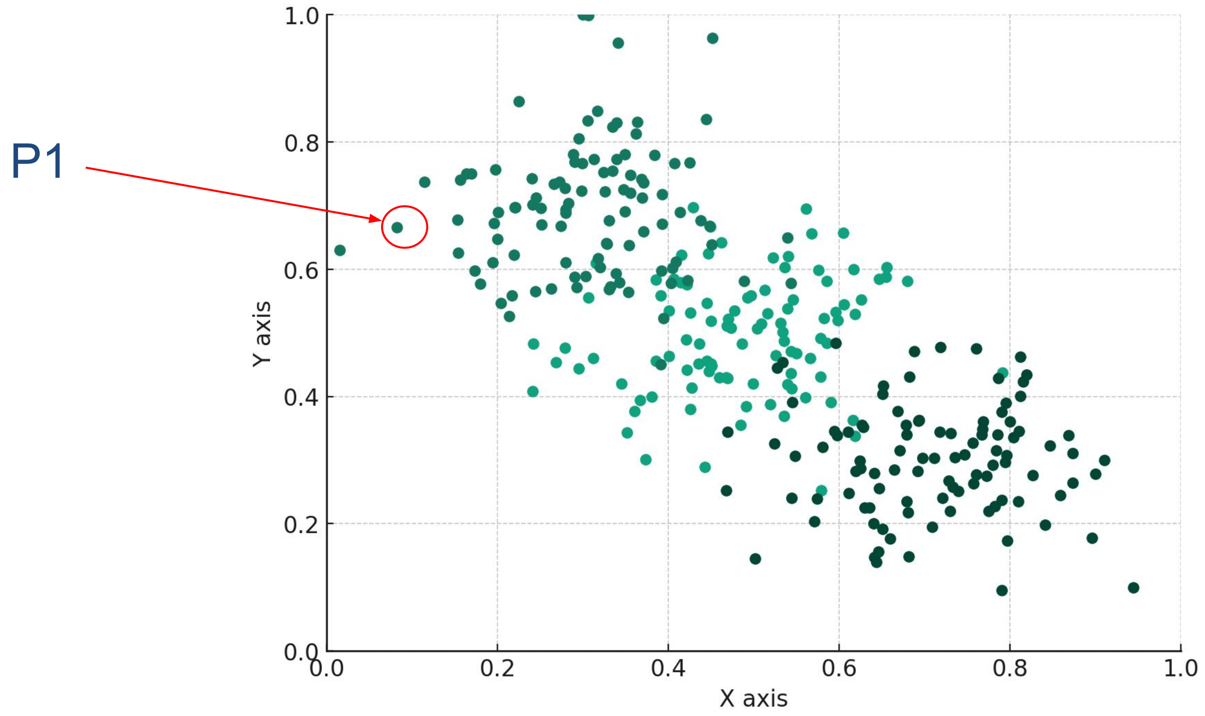
What is a Vector Embedding?



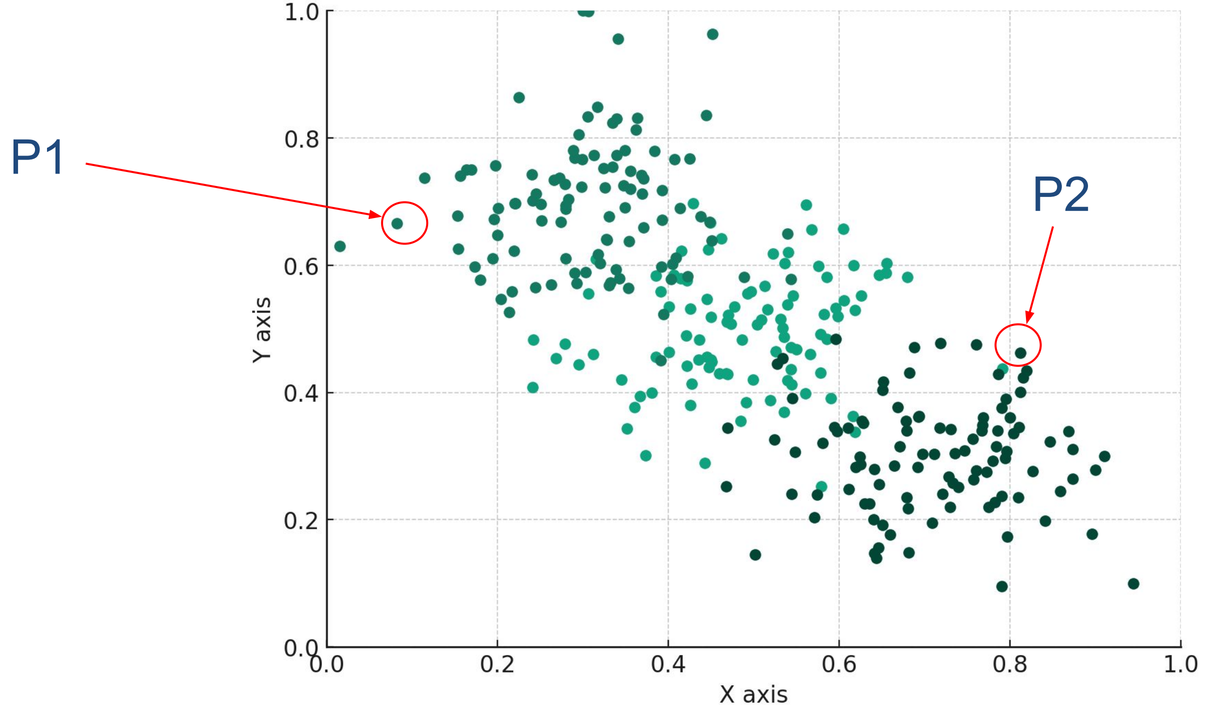
2D example



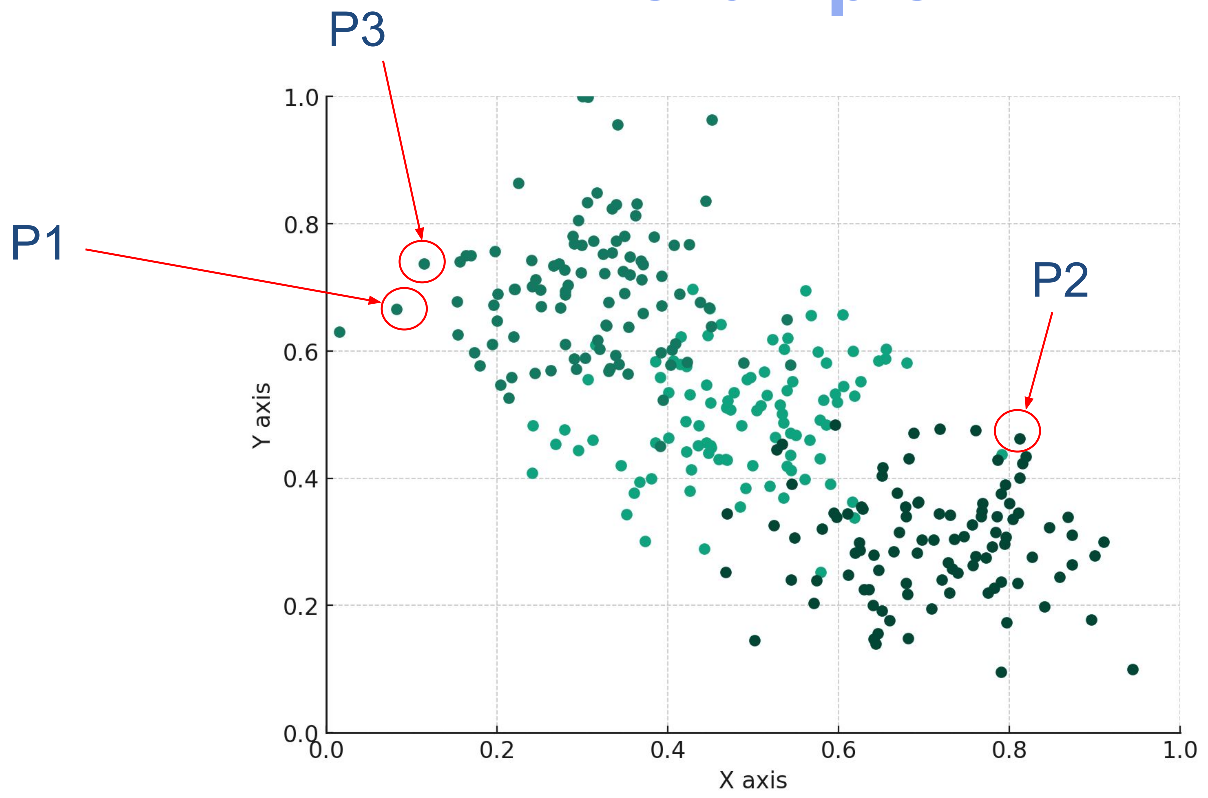
2D example



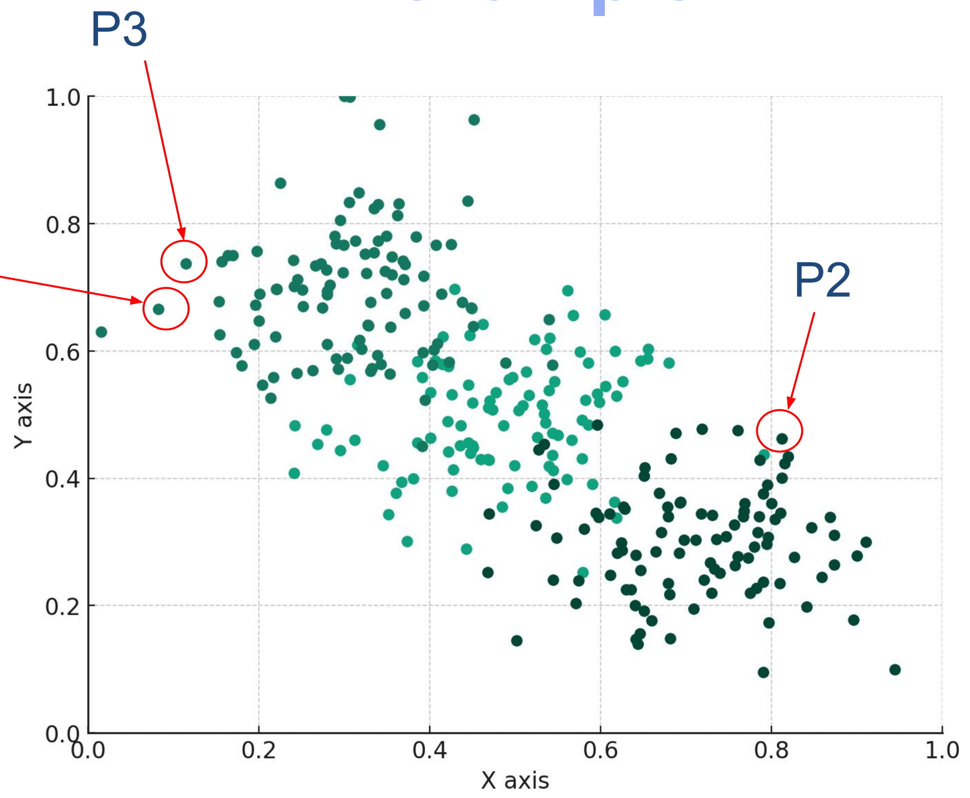
2D example



2D example



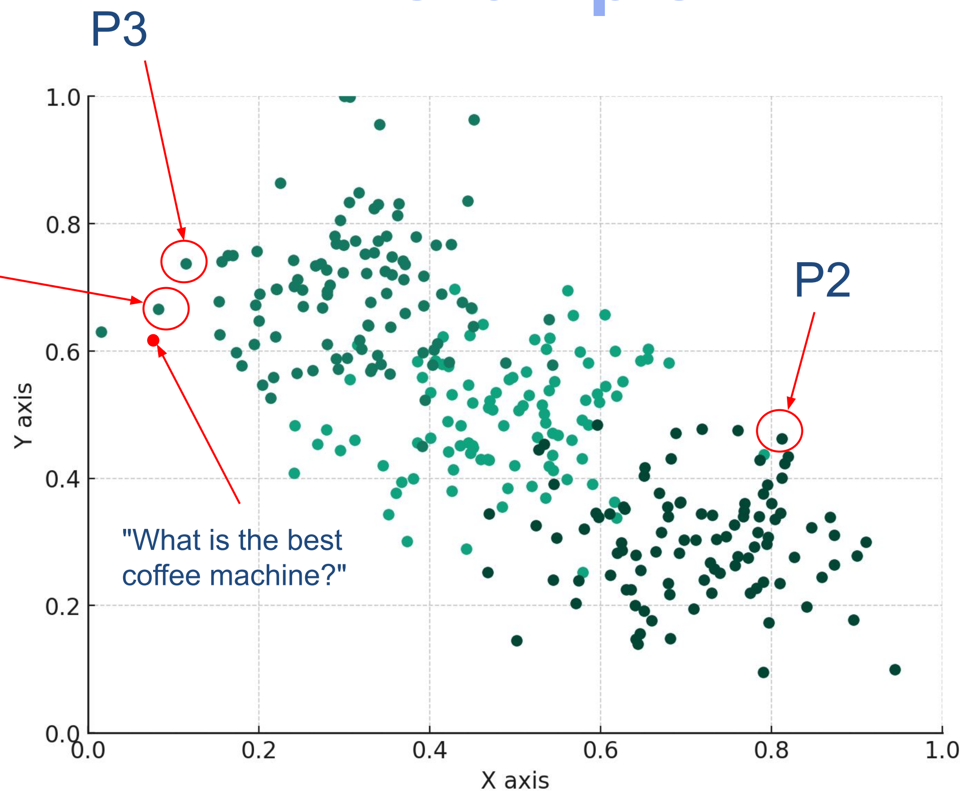
2D example



The document associated with P1 is more similar to P3 than P2,

according to the model that generated the points!

2D example



The document associated with P1 is more similar to P3 than P2,

according to the model that generated the points!

As a database user, what must you do?

1. Install a vector database (MariaDB Vector preview now available)

2. Install an Embedding Model
   or
   Setup a cloud hosted model API.

3. Change your application to query the Embedding Model for each document insert and insert the embeddings into the database.

4. Make use of VEC_DISTANCE function to get the (approximate) nearest neighbors.

Create an embedding index

PRODUCTS		
NAME	DESCRIPTION	EMBEDDING
"Coffee Maker"	"Can brew 10 different coffee types. 5 years warranty."	[0.4, 0.5, 0.3, ….., 0.2]

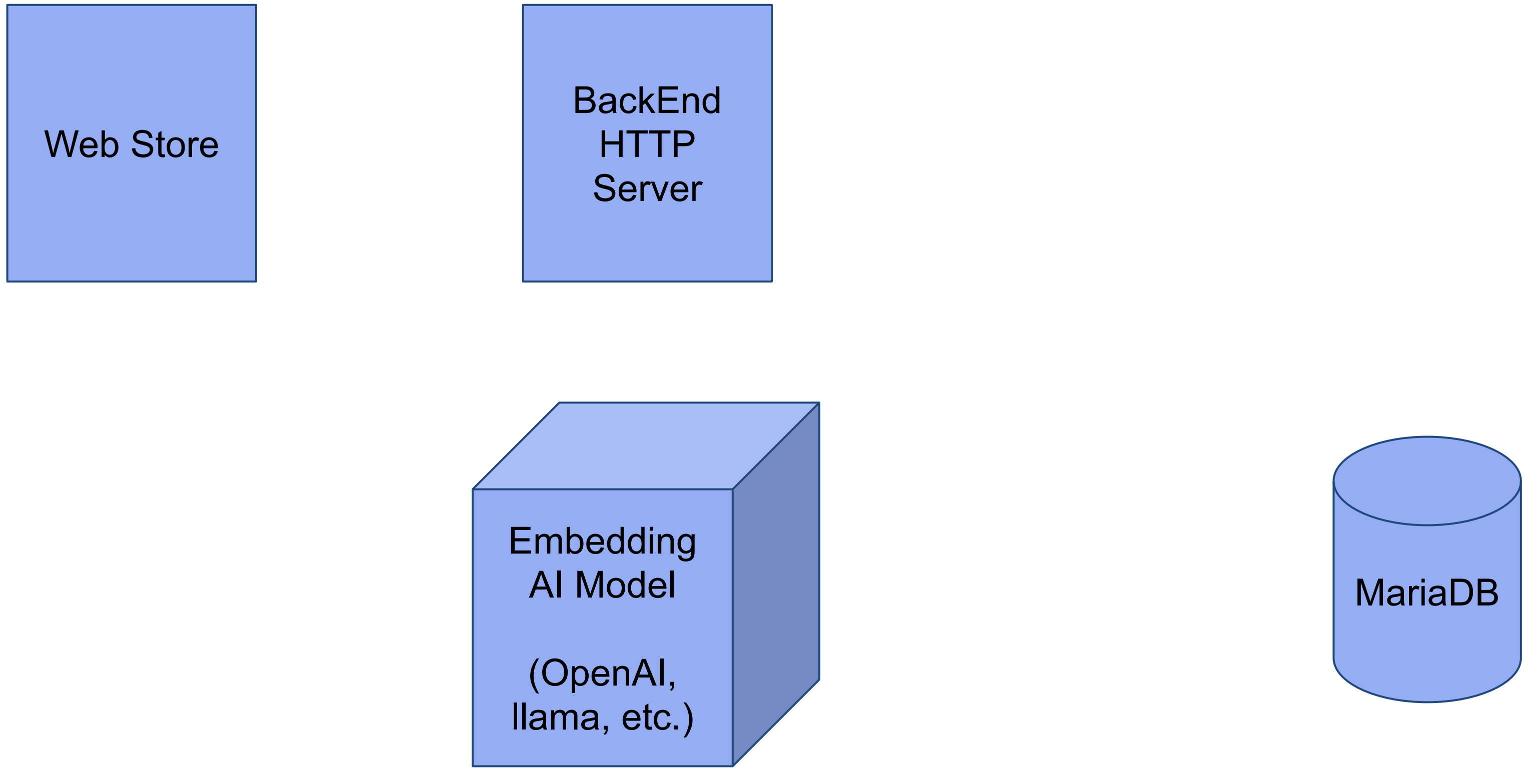
```
CREATE TABLE PRODUCTS (
    name varchar(200) primary key,
    description longtext,
    embedding blob,
    VECTOR INDEX (embedding) MHNSW_M=5
)
```

Modify insert

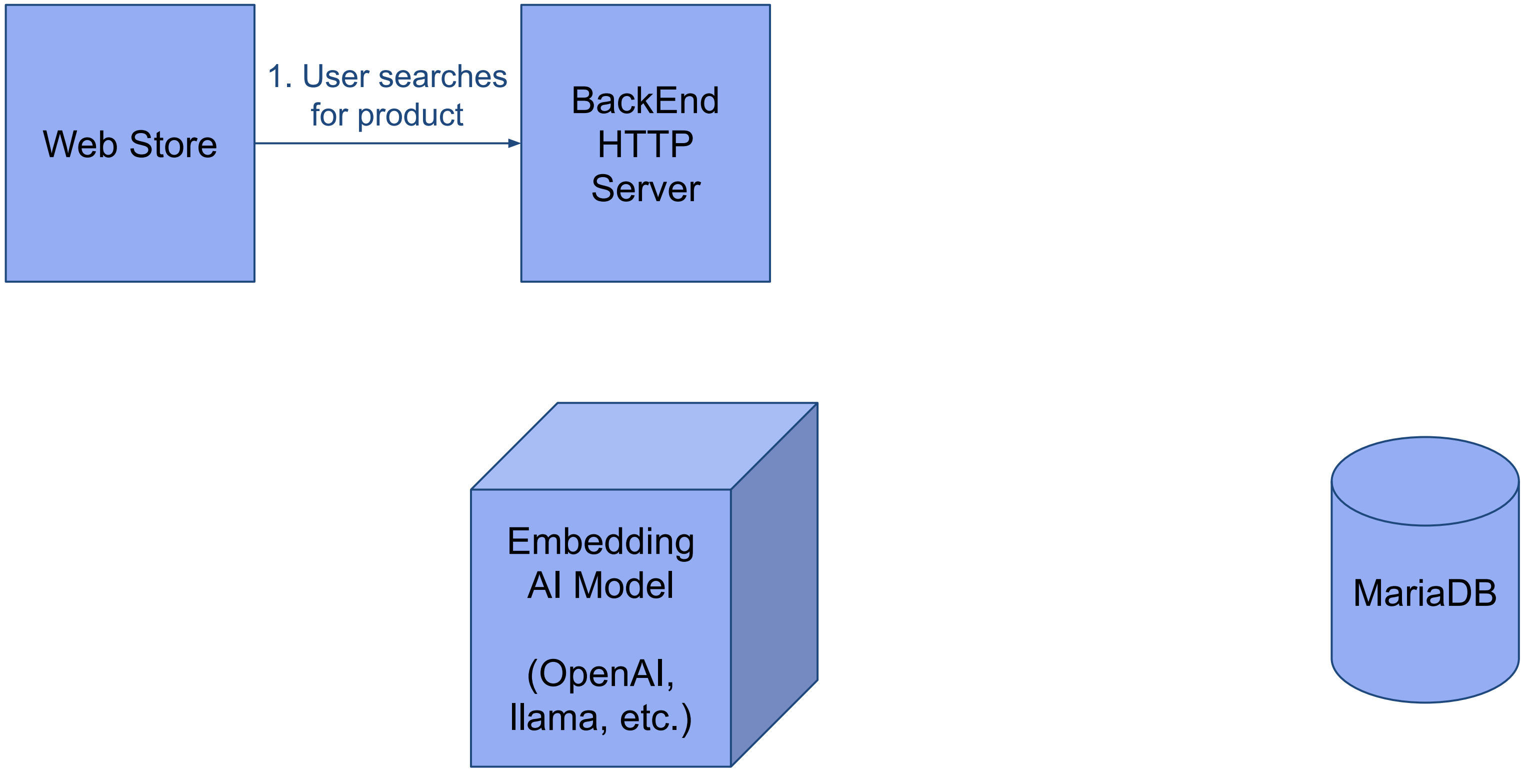
PRODUCTS		
NAME	**DESCRIPTION**	**EMBEDDING**
"Coffee Maker"	"Can brew 10 different coffee types. 5 years warranty."	[0.4, 0.5, 0.3, ….., 0.2]

```
def add_product(db_conn, product, ai_model):
  vector = ai_model.get_embedding(product.name +
                                   product.description)
  db_conn.execute(
      'INSERT INTO products (name, description, embedding)'
      'values (?, ?, ?)',
      (product.name, product.description, vector))
```

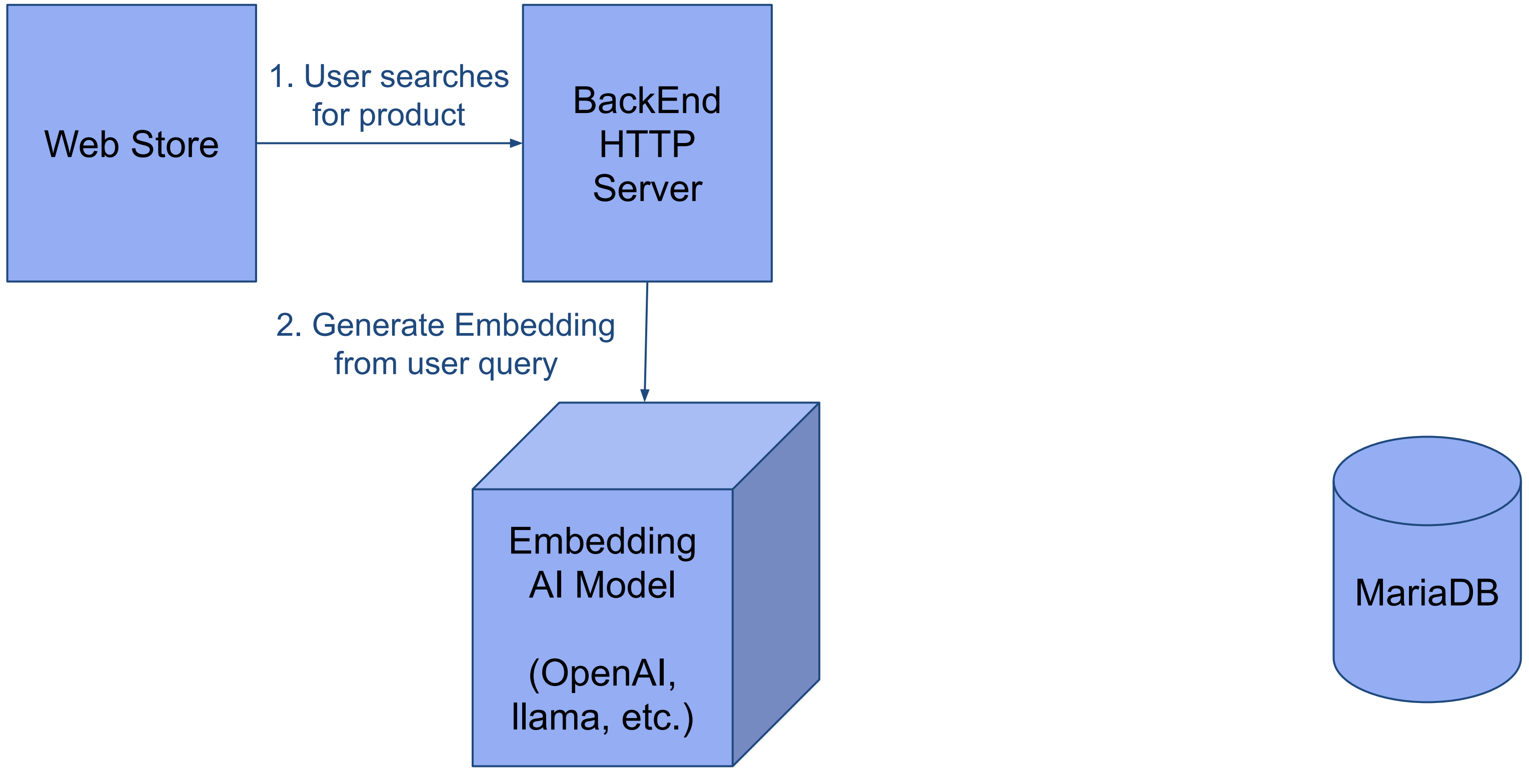
Where Vector search comes into play?



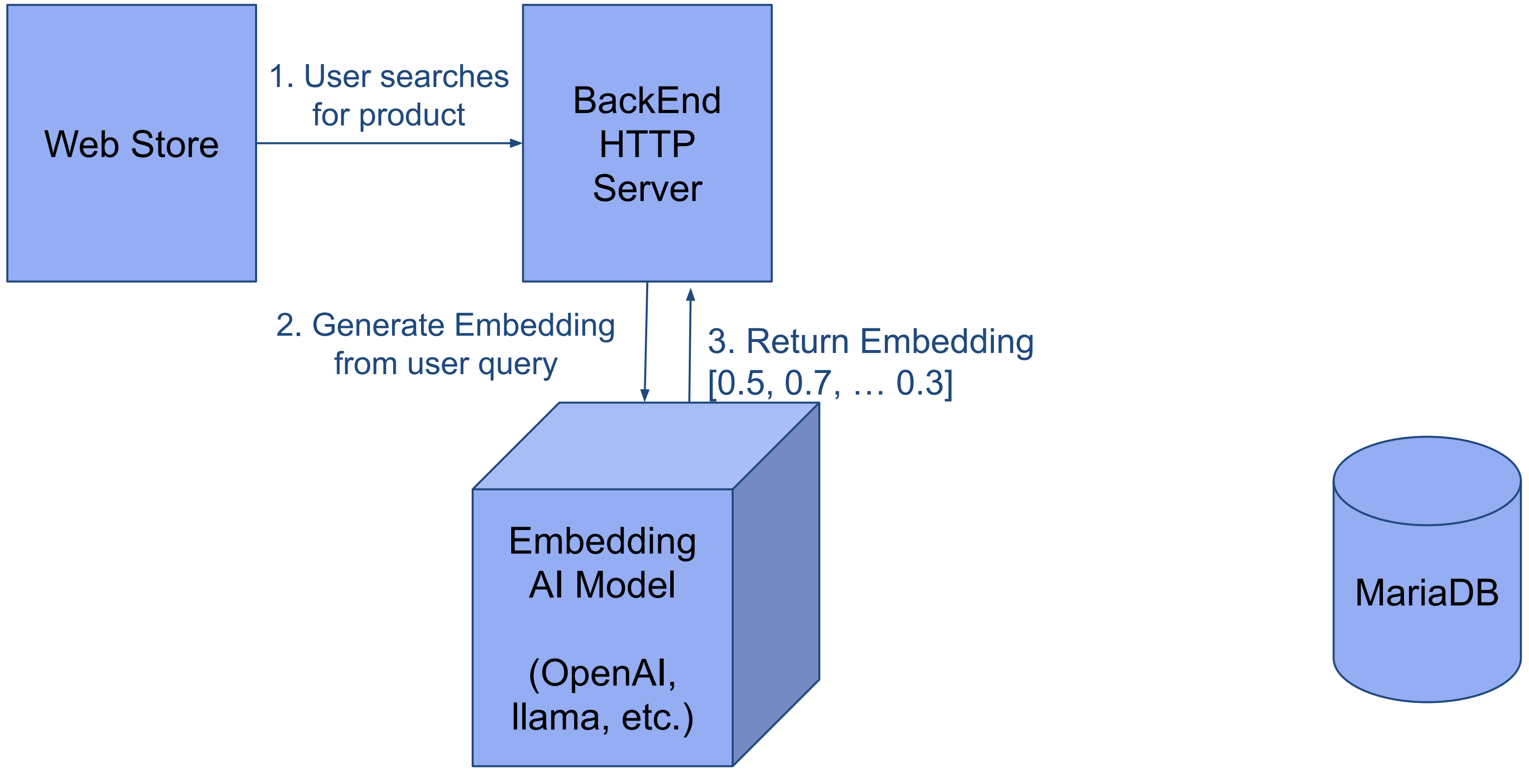
Where Vector search comes into play?



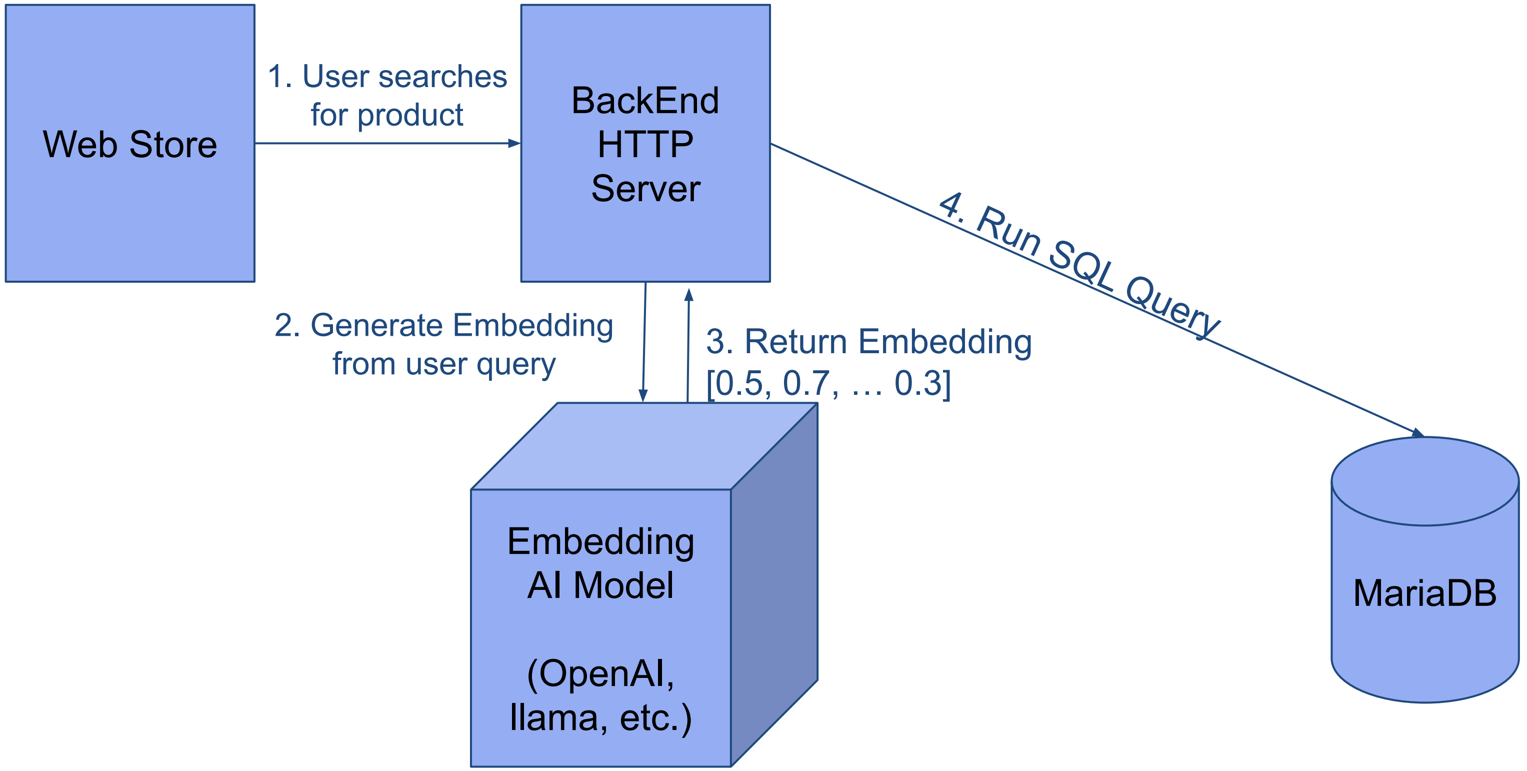
Where Vector search comes into play?



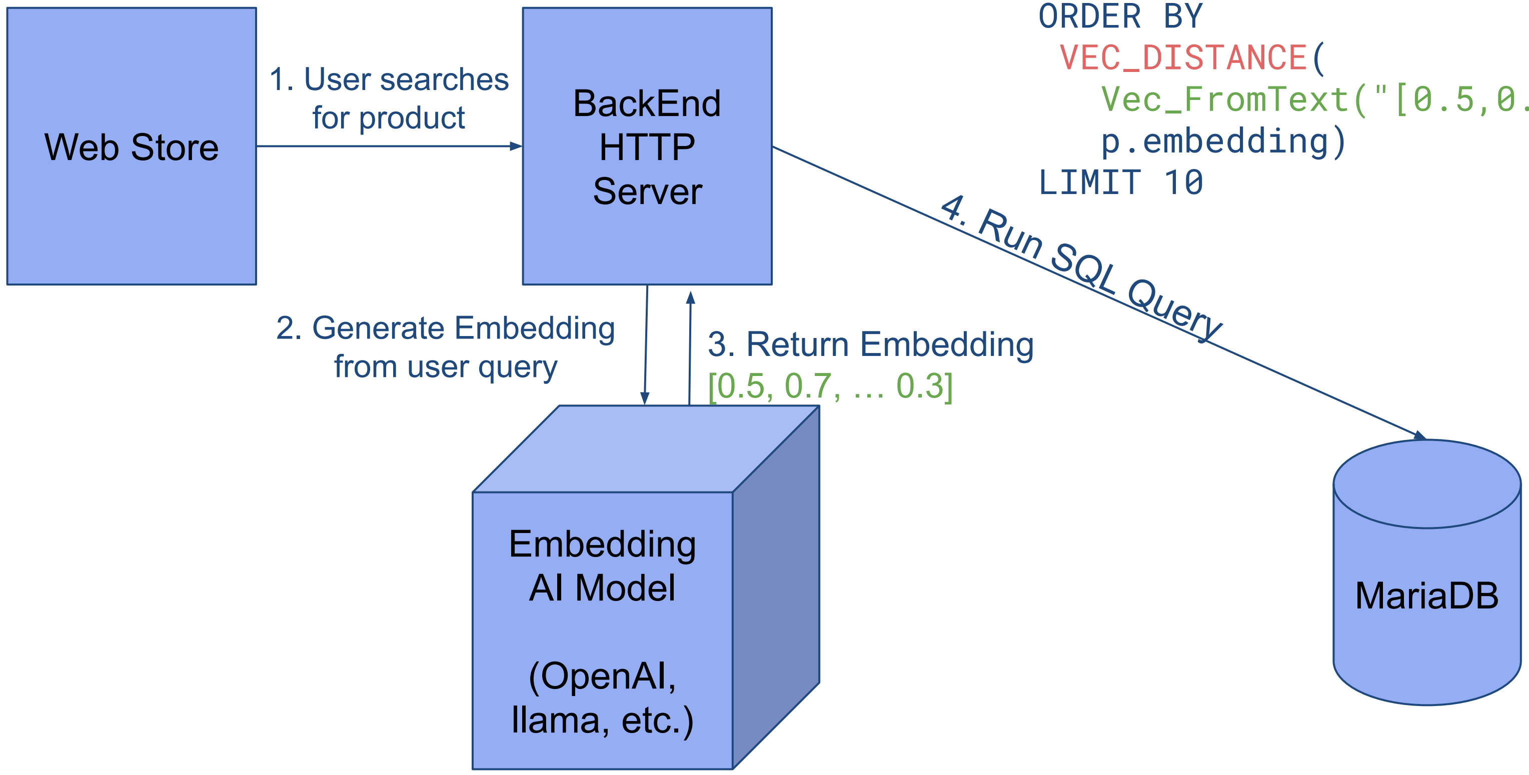
Where Vector search comes into play?


Web Store
1. User searches for product
BackEnd HTTP Server
2. Generate Embedding from user query
3. Return Embedding [0.5, 0.7, … 0.3]
Embedding AI Model (OpenAI, llama, etc.)
MariaDB

Where Vector search comes into play?

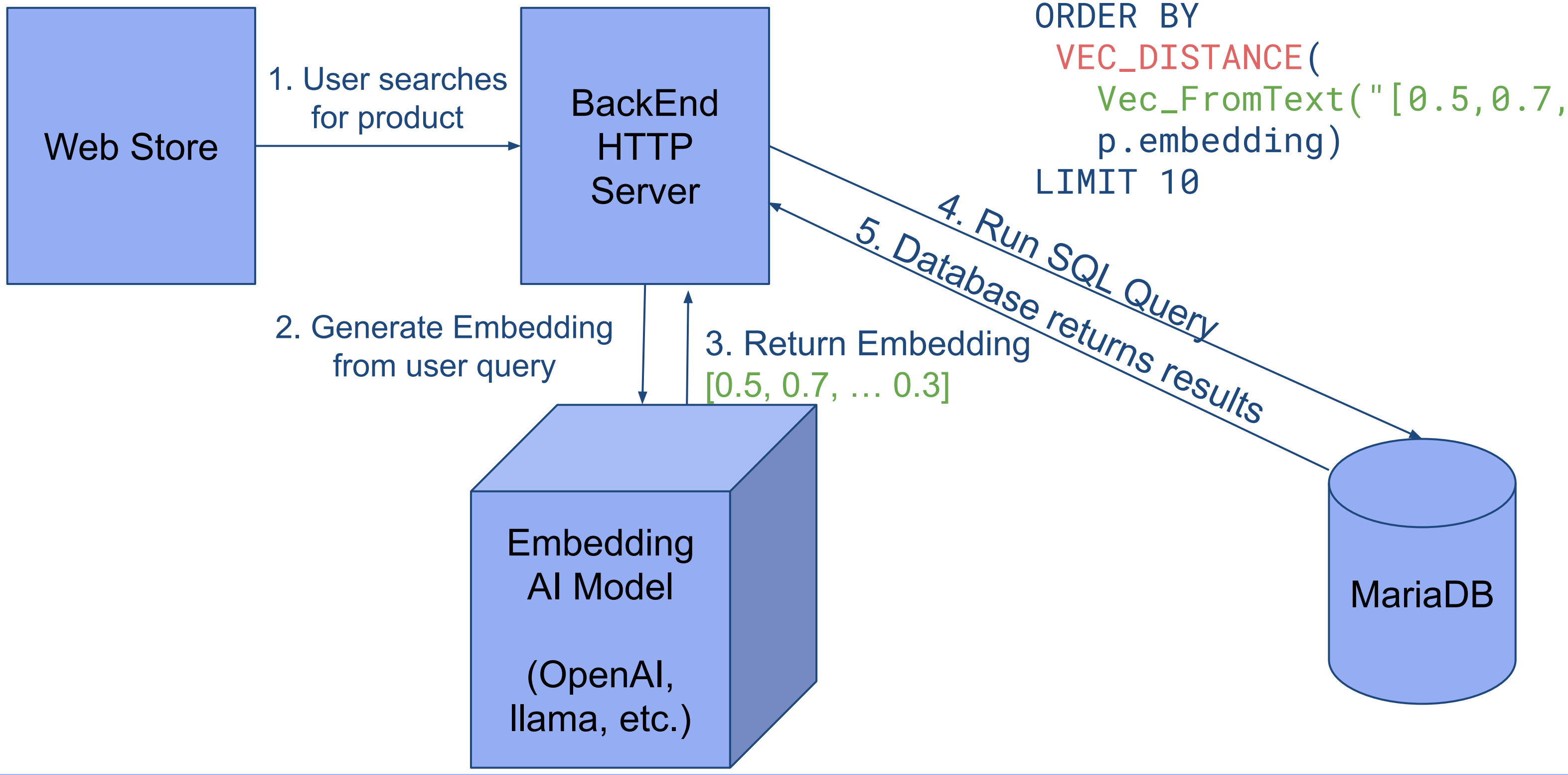

Web Store
1. User searches for product
BackEnd HTTP Server
2. Generate Embedding from user query
3. Return Embedding [0.5, 0.7, … 0.3]
4. Run SQL Query
Embedding AI Model
(OpenAI, llama, etc.)
MariaDB

Where Vector search comes into play?



```
SELECT p.name, p.description
FROM products p
ORDER BY
 VEC_DISTANCE(
   Vec_FromText("[0.5,0.7,…,0.3]"),
   p.embedding)
LIMIT 10
```

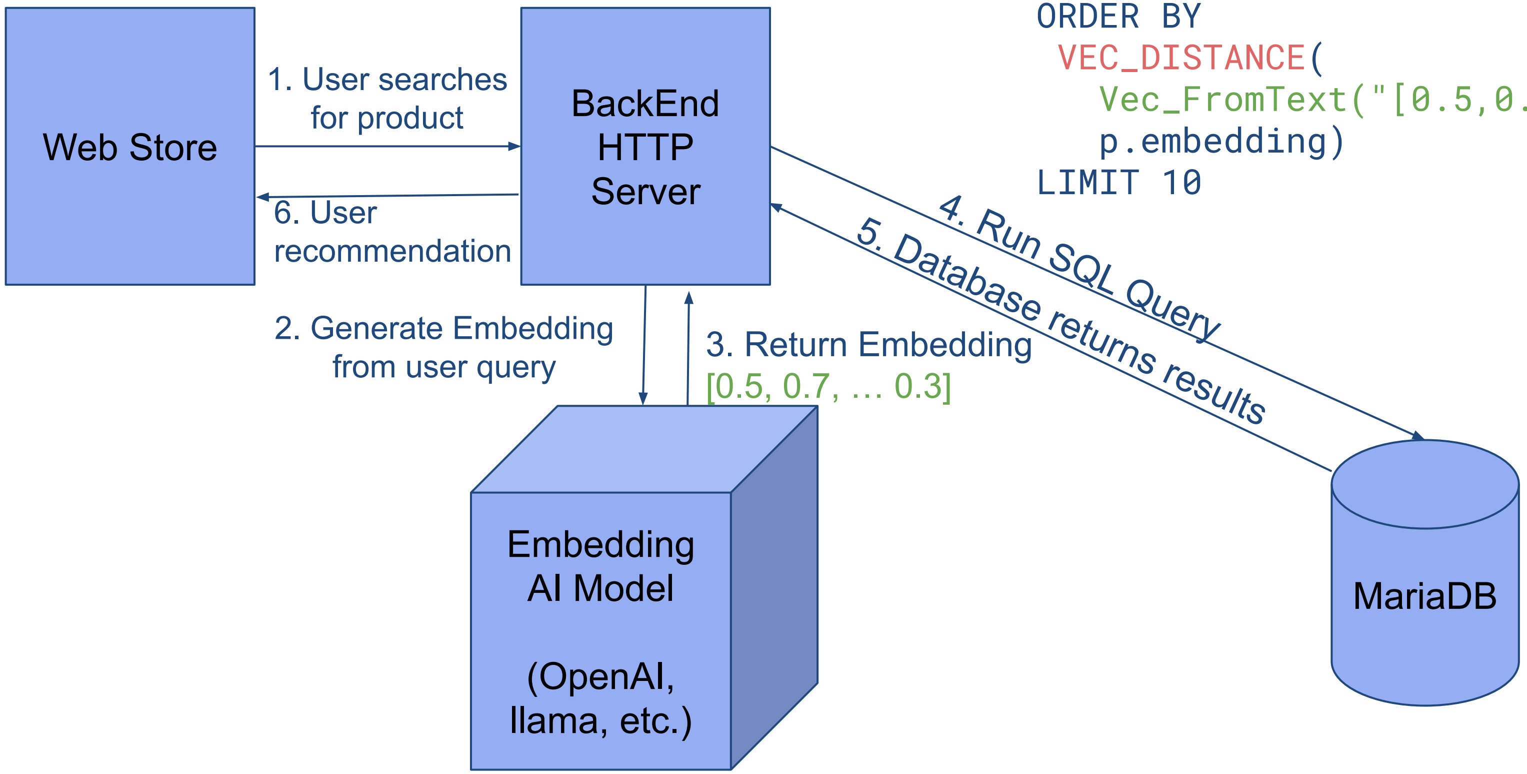
Where Vector search comes into play?



```
SELECT p.name, p.description
FROM products p
ORDER BY
 VEC_DISTANCE(
   Vec_FromText("[0.5,0.7,…,0.3]"),
   p.embedding)
LIMIT 10
```

Where Vector search comes into play?

```
SELECT p.name, p.description
FROM products p
ORDER BY
 VEC_DISTANCE(
   Vec_FromText("[0.5,0.7,…,0.3]"),
   p.embedding)
LIMIT 10
```



What's the catch?

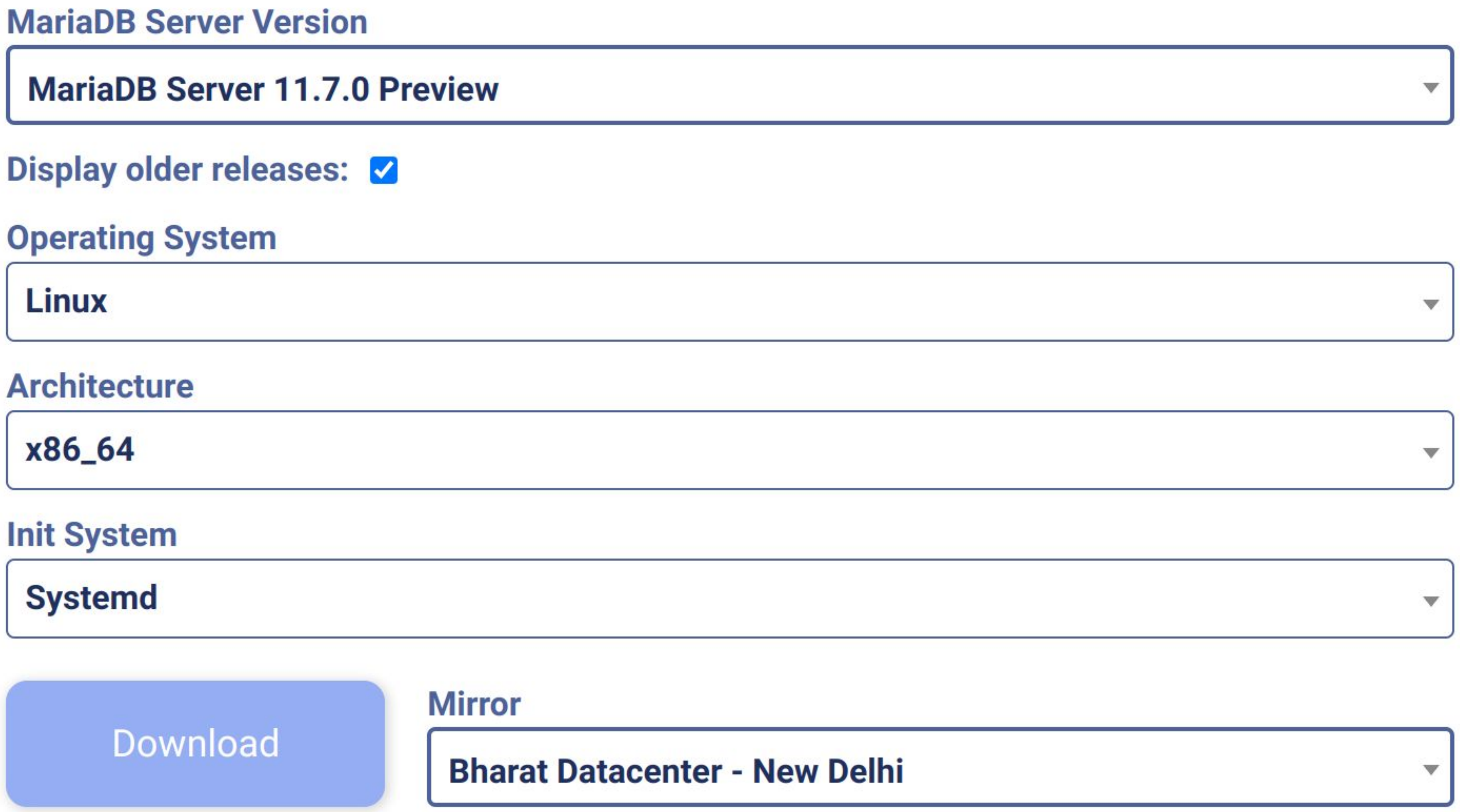
1. Searching for vectors is expensive
2. Indexing strategies for vectors are only "approximate", they don't guarantee the exact "nearest" neighbour.
3. Depending on dataset, some indexing strategies perform better than others.
4. Indexing generally requires a lot of memory.
   a. **HNSW – Hierarchical Navigable Small Worlds**
      i. **de-facto industry standard. Implemented in MariaDB**
      ii. **Large memory usage.**
   b. IVFFlat – Low resource usage, poor search quality, present in pgvector

Project status

- 11.7 is first stable release (MDEV-33408)
- Performance faster than pgVector on SELECTS (better scaling)
  - More optimizations planned (ARM, PowerPC operations).

- Preview of MariaDB Vector syntax supports:
  - VEC_DISTANCE
  - VEC_DISTANCE_COSINE (euclidean / cosine distance)
  - VEC_FromText() VEC_ToText()

- Work collaboratively with MariaDB plc and other vendors (large contributions from Amazon)



MariaDB Server Version

MariaDB Server 11.7.0 Preview

Display older releases: ✓

Operating System

Linux

Architecture

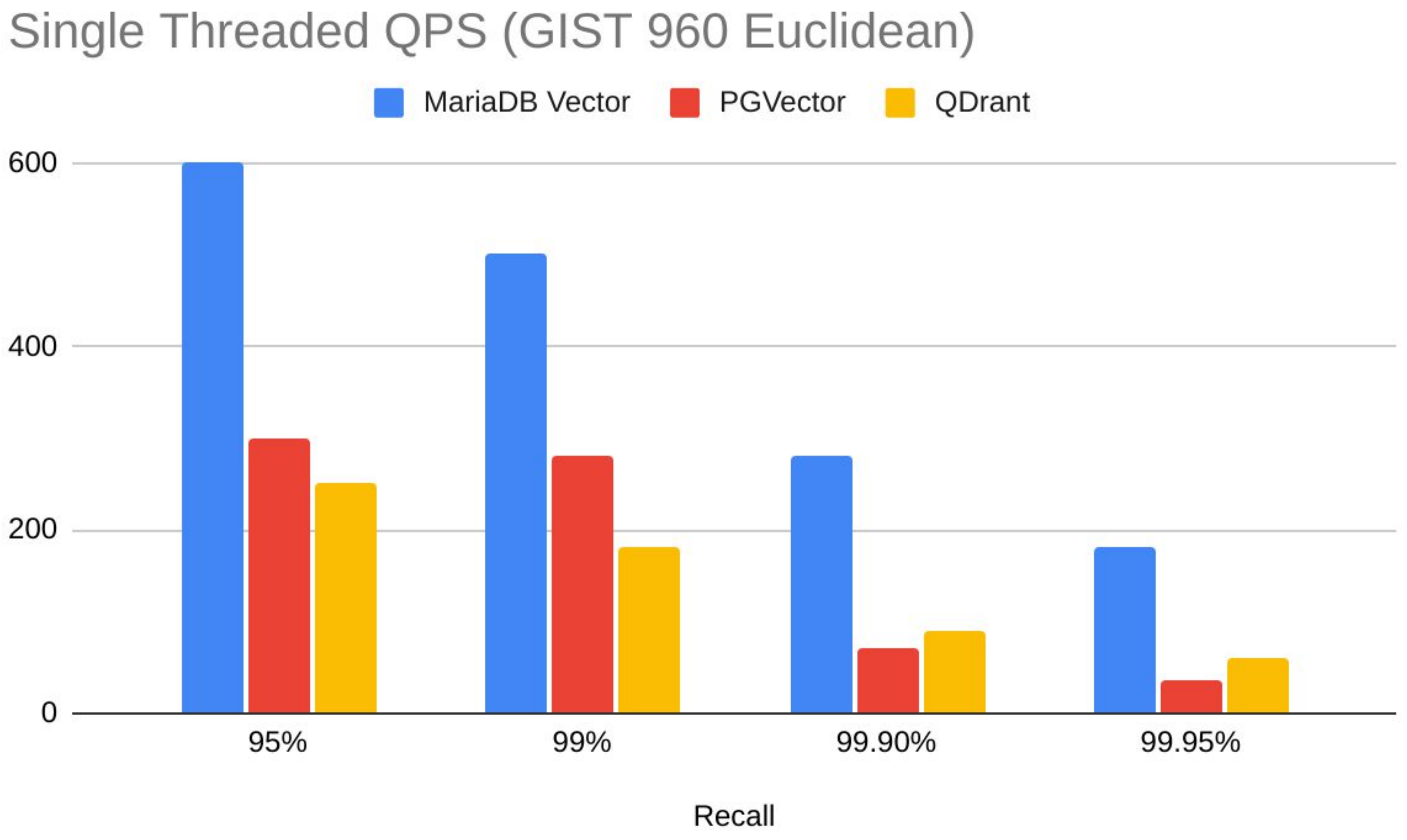
x86_64

Init System

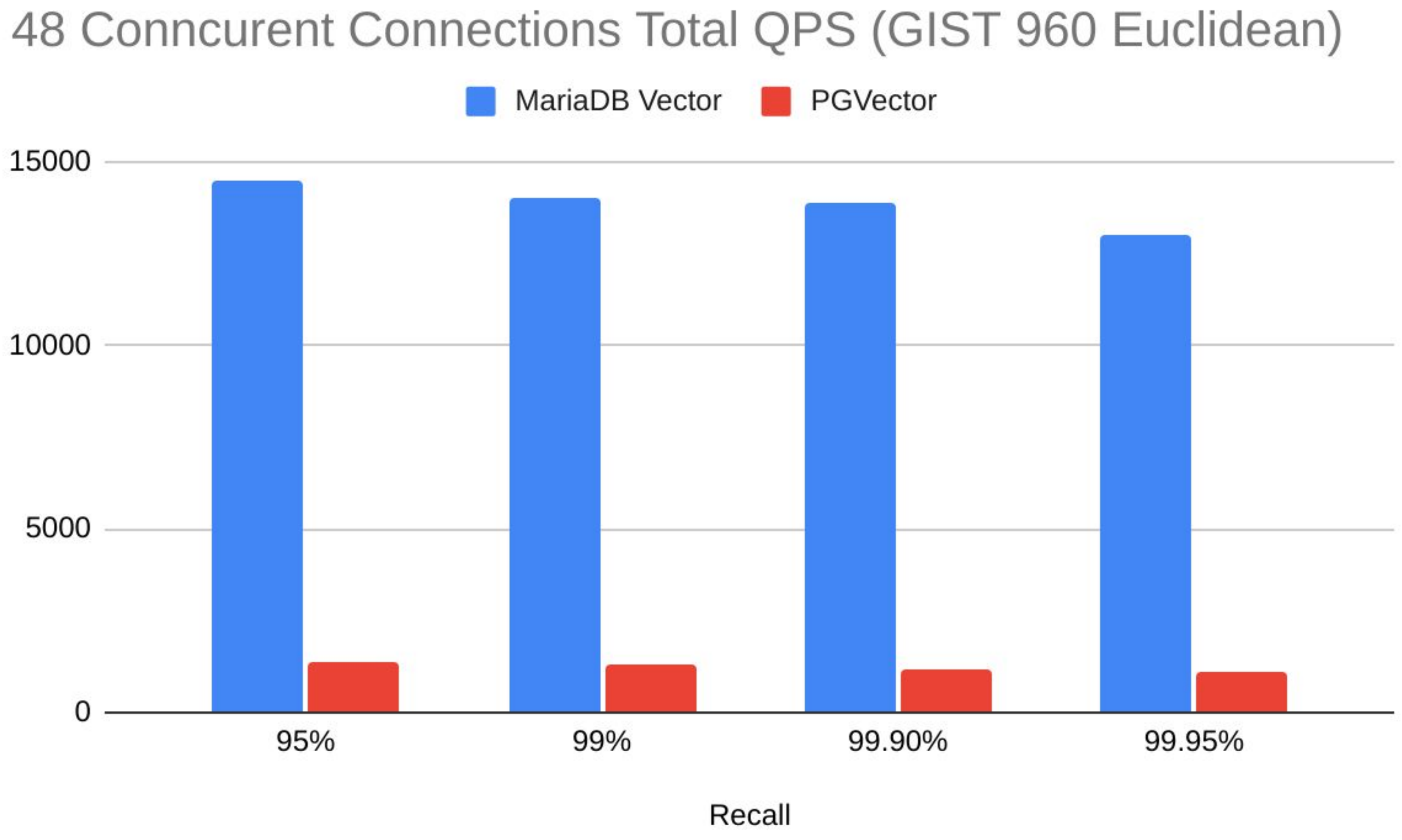
Systemd

Mirror

Bharat Datacenter - New Delhi



https://mariadb.com/es/resources/blog/how-fast-is-mariadb-vector/



https://mariadb.com/es/resources/blog/how-fast-is-mariadb-vector/

Index Construction

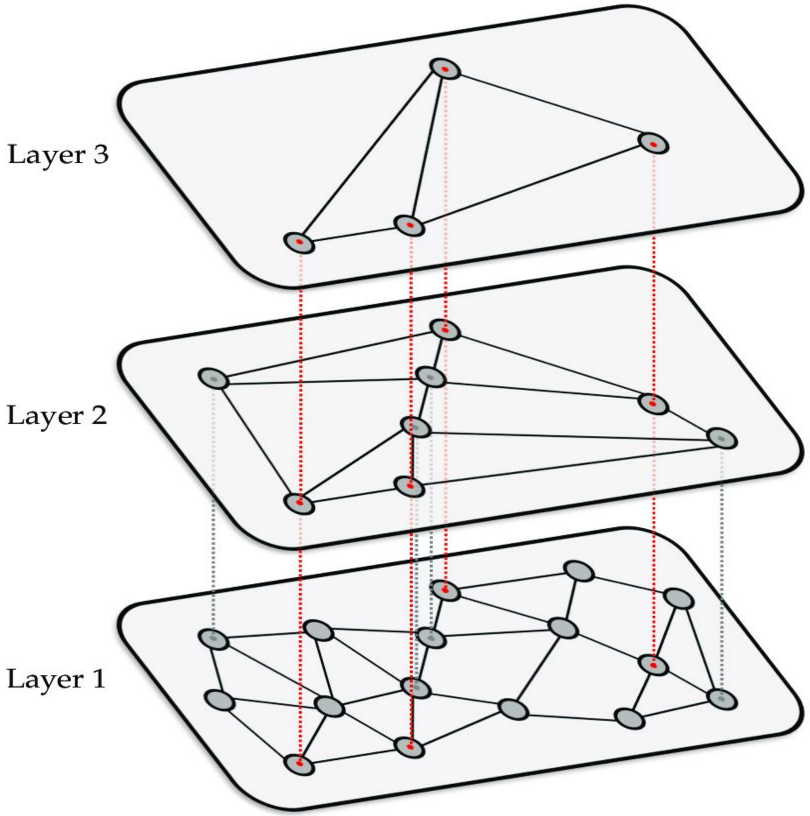
1. HNSW index is stored as a separate auxiliary table

   [**layer**, **tref**, vec, neighbors]

2. ACID benefits of the underlying storage engine.

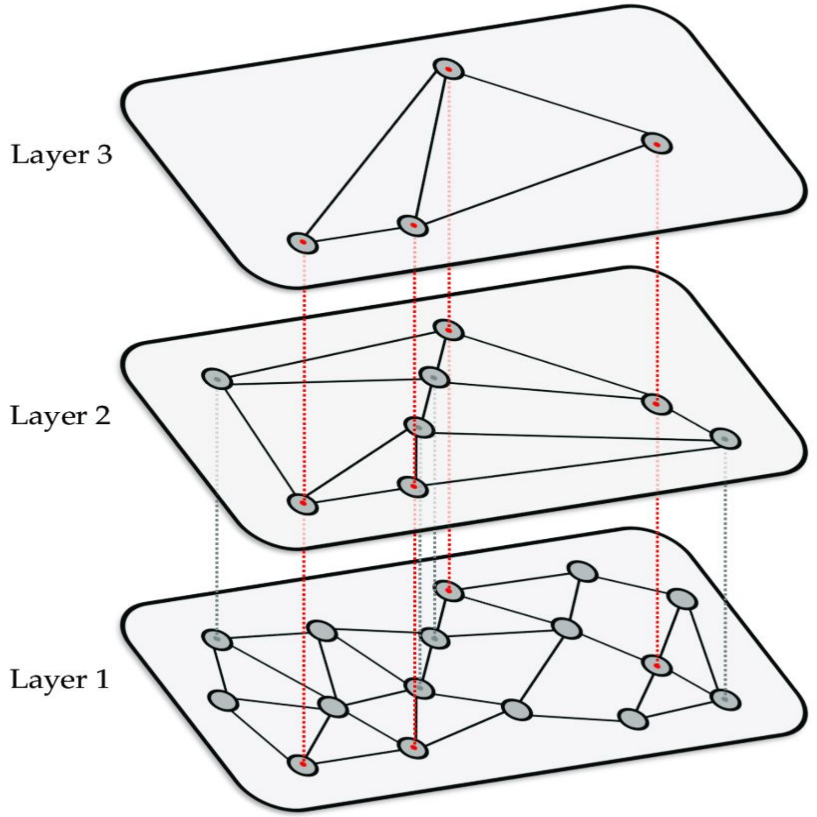
3. A bit more overhead than having it natively within the SE.

4. More flexible.



Index Construction

1. HNSW allows online construction.
2. HNSW does not have a native DELETE method.
3. Parameters that influence index quality / speed:
   a. **mhnsw_max_edges_per_node** (**5-8** is ok)



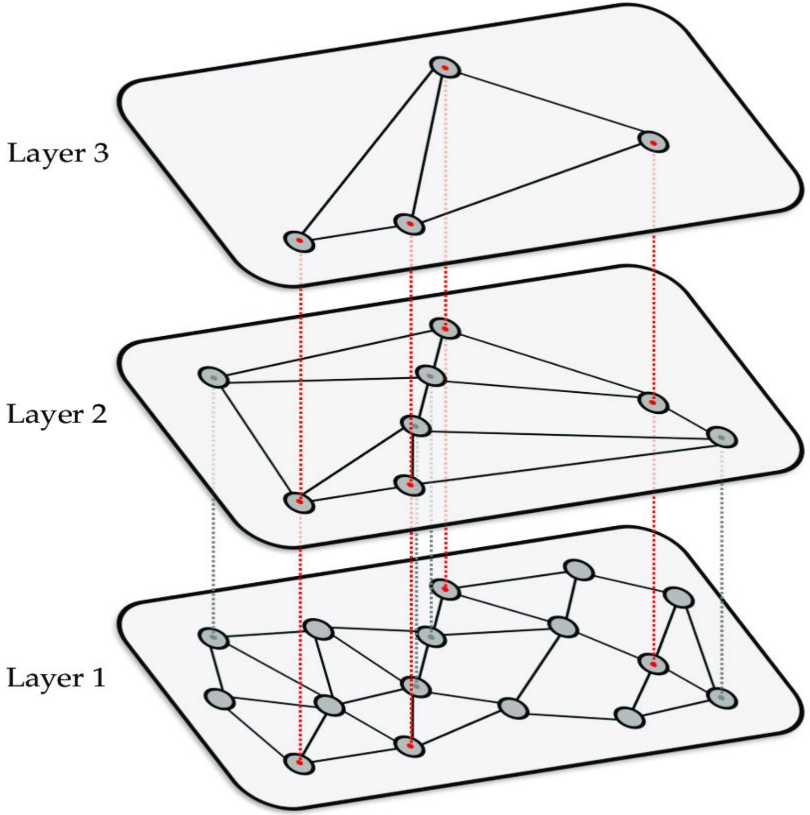
Index Lookup

1. Traverse the graph from upper layer to lower layer.

2. Parameters that influence results:

3. **mhnsw_ef_search** from HNSW paper

4. Higher values produce better recall.
   (percentage of results which are true minimums)

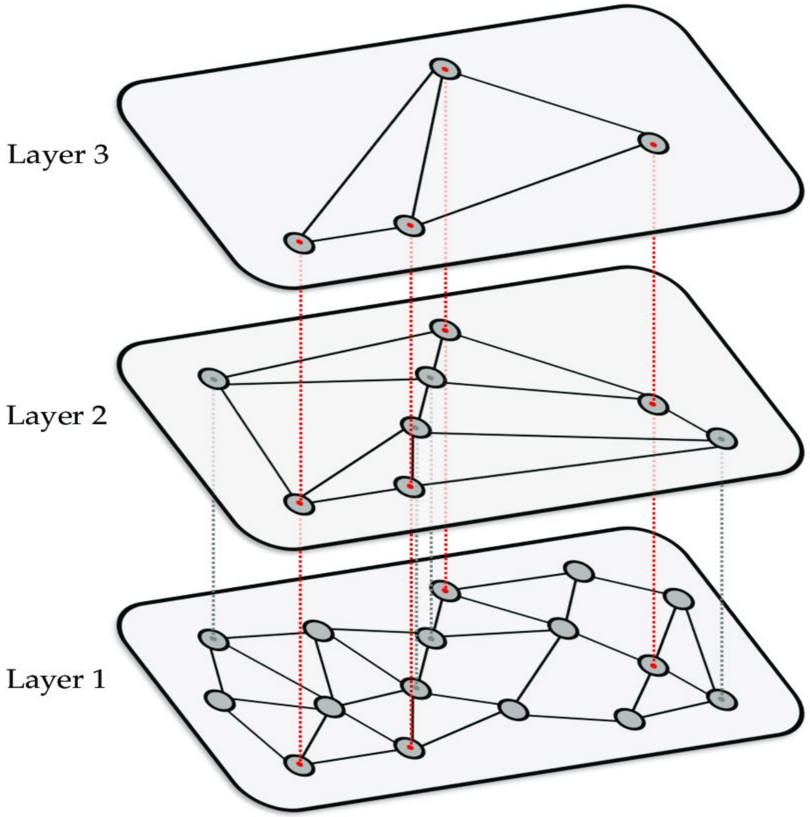


Index Lookup

1. MariaDB has a dedicated shared-statement cache to store the graph in memory.

   mhnsw_cache_size

   Ideally this should fit all your vector data for best performance.



Possible future directions?

1. Plugins to generate embedding on insert.

2. Storage Engine for Vector Embeddings generation (CONNECT SE can fulfill this to some degree already)

3. More vector indexing algorithms.
   a. IVFFlat is a Google Summer of Code project this year.

4. Performance optimizations - Index Condition pushdown

Thank you!

Contact details:

kaj@mariadb.org

vicentiu@mariadb.org

About:

https://mariadb.org/projects/mariadb-vector/

https://mariadb.org/kaj

https://mariadb.org/vicentiu